Robust Partitioning for Real-Time Multiprocessor Systems with Shared Resources

Frédéric Fauberteau, Serge Midonnet

Robust Partitioning for Real-Time Multiprocessor Systems with Shared Resources

Frédéric Fauberteau
Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne
77454 Marne-la-Vallée CEDEX 2, FRANCE
frederic.fauberteau@univ-paris-est.fr

Serge Midonnet
Université Paris-Est
LIGM, UMR CNRS 8049
5, bd Descartes, Champs-sur-Marne
77454 Marne-la-Vallée CEDEX 2, FRANCE
serge.midonnet@univ-paris-est.fr

ABSTRACT

In this paper, we focus on the temporal robustness in the hard real-time multiprocessor systems. This robustness is the capacity to tolerate faults in such a way that no deadlines are missed. A model of sporadic and dependent tasks is considered. Our contribution is to propose a partitioning algorithm which assigns the tasks to processors in order to maximize the robustness of the system to Worst Case Execution Time (WCET) overruns faults or Minimum Inter-arrival Time (MIT) violations.

Keywords

Real-time Scheduling, Multiprocessor Systems, Robustness, Partitioning Algorithms, Shared Resources.

1. INTRODUCTION

We consider that an application consists of a set of tasks where each task is recurring jobs. One of the main problems in real-time scheduling is to guarantee that no task misses its deadline. Uniprocessor scheduling attempts to solve the *priority problem*: when, and in what order should each job execute. In multiprocessor scheduling, to the *priority problem* is added the *allocation problem*: on which processor a job should execute. Multiprocessor scheduling algorithms can be classified according theses two problems.

We focus on the *static-priority* approach (for the *priority problem*) for which each task has a single static priority applied to all of its jobs and on the *partitioned* approach (for the *allocation problem*) for which each task is allocated on a processor and no migration is allowed. The partitioned scheduling approach consists in partitioning a taskset among processors. Each subset of task is scheduled on in own processor independently of others. The *partitioning problem* can be seen as BIN-PACKING *problem* which is NP-hard in the strong sense. Hence no optimal algorithm exists unless $P = NP$. But several approaches are inspired by techniques for BIN-PACKING and enable to solve partitioning problem.

RACS 2011 November 2-5, 2011, Miami, FL, USA.

We also consider that the tasks are dependent in the sense that they share resources. Then it is necessary to use a synchronization protocol to avoid unbounded blocking times due to priority inversion. In this context, we are interested in supplying robustness by maximizing the capacity of tasks to stand up to WCET overruns faults. The robustness is the capacity to tolerate temporal faults such as WCET overruns or MIT violations. When a temporal fault occurs (due to a high priority task), the low priority and non faulty tasks are protected against deadline missed (temporal failure). This property is called the fault isolation.

1.1 Related work

One of the most prevalant strategies to provide fault tolerance is replication. Qin and Hong has proposed an algorithm which generates distributed static schedules to handle processor failures [14]. Emberson and Bate have proposed a task allocation algorithm which takes into account static redundancy to dispatch the tasks among the processors [10]. Our paper is concerned by a strategy of margin on the temporal constraints of the tasks. Instead to replicate tasks to deal with faults, we give them the possibility to continue their execution beyond their WCET.

Echtle and Eusgeld used a genetic algorithm to find fault-tolerant partitions [9]. However, their work does not focus on real-time systems. Tindell, Burns and Wellings have proposed a partitioning algorithm based on simulated annealing to allocate tasks on a multiprocessor distributed systems [17]. Di Natale and Stankovic have also used this technique to deal with systems of tasks with jitter [8]. These two works are aimed at real-time systems but the fault-tolerance is not taken into account.

Oh and Son discuss the need to consider schedulability and fault-tolerance simultaneously [13]. They prove that finding a schedule to handle a single processor failure is a NP-hard problem. Their model does not include dependency constraints.

In this paper, we propose a partitioning algorithm which is fault-tolerant in such a way that it dispatches the tasks in order to maximize the margin (on WCET overruns or on frequency increasing).

1.2 Terminology

We consider a set $\tau = \{\tau_1, \tau_2, \ldots, \tau_n\}$ of n sporadic real-time tasks. A sporadic task is a recurring task for which only a lower bound on inter-arrival time of the jobs is known. Each task τ_i is characterized by a minimum inter-arrival time T_i (also denoted period), a WCET C_i and a relative deadline D_i. The considered tasks are tasks with constrained deadlines (i.e. $D_i \leq T_i$). The processor utilization of τ_i is denoted U_i and is defined as $U_i = C_i/T_i$. This application runs on a platform $\Pi = \{\pi_1, \pi_2, \ldots, \pi_m\}$ of m identical processors (homogeneous case). A job J_i is characterized by its release time r_i, its execution time e_i (observed at run time) and its absolute deadline d_i. The processor utilization of the τ is denoted $U(\tau)$ and the total processor utilization of π_j is denoted $U(\pi_j)$. We consider a static-priority scheduling on each processor. The priorities of the tasks are taken in an increasing order (*i.e.* the priority of τ_{i+1} is higher than the priority of τ_i). The set of tasks with a priority lower (resp. higher) than the priority of τ_i is denoted $lp(\tau_i)$ (resp. $hp(\tau_i)$). In this paper, we denote R_i the response time of a task τ_i and B_i the blocking factor incurred by the task τ_i. We also denote A_i^C (resp. A_i^f) the value of WCET (resp. frequency) margin of τ_i.

1.3 Organization

The rest of this paper is organized as follows. In Section 2 we present several synchronization protocols for multiprocessor systems. In Section 3, we present the concept of robustness to WCET overruns faults and MIT violations. We describe how to compute the margin of the tasks on their WCET and their period. In Section 4, we propose a partitioning algorithm which maximize the robustness in terms of WCET overruns faults. In Section 5, we compare by simulation our approach with existing ones. Finally, we conclude and give some perspectives in Section 6.

2. SYNCHRONIZATION PROTOCOLS

In real-time systems, the synchronization protocols are used to avoid the unbounded priority inversions which could occur with a simple lock mechanism. In the multiprocessor context, three main protocols are often cited in the literature. We present them in the following sections.

2.1 MPCP

Multiprocessor Priority Ceiling Protocol (MPCP) is an extension to the multiprocessor case of the *Priority Ceiling Protocol* (PCP) proposed by Sha, Rajkumar and Lehoczky in [16]. MPCP has been proposed by Rajkumar in [15].

With PCP, a priority ceiling $\overline{p}(\mathcal{R}_k)$ is associated with each shared resource $\mathcal{R}_k$. $\overline{p}(\mathcal{R}_k)$ is defined as the priority of the highest priority job which can lock $\mathcal{R}_k$. At time t, the system priority ceiling $\overline{p}(t)$ is given by the highest priority ceiling of the locked resources at t. A job J_i can lock a resource $\mathcal{R}_k$ at time t only if the priority of J_i is strictly greater than the system priority ceiling. Otherwise, the job which blocks J_i inherits the priority of J_i.

With MPCP, two types of resources are distinguished: local and global resources. The local resources are only shared by jobs on the same processor whilst global ones are shared by jobs which can be assigned on several processors. When a job tries to lock a local resource, the behavior of MPCP is the same as PCP. For the case of global resources, a set of priorities higher than the highest priority ceiling is used. Then, accesses to global resources are made uppermost to limit indirect blocking.

2.2 MSRP

Multiprocessor Stack Resource Protocol (MSRP) is an extension to the multiprocessor case of the *Stack Resource Protocol* (SRP) proposed by Baker in [1]. MSRP has been proposed by Gai *et al.* in [11].

With SRP, each job J_i is characterized by a preemption level $\rho(J_i)$. J_i is allowed to preempt J_j only if $\rho(J_i) > \rho(J_j)$. Each resource $\mathcal{R}_k$ has a preemption ceiling $\rho(\mathcal{R}_k)$ defined as the highest preemption level of the jobs which can lock $\mathcal{R}_k$. At time t, the system preemption ceiling is given by the highest preemption ceiling among resources currently locked. The activ jobs are stored in a stack in decreasing order of their priority. J_i is allowed to preempt J_j if J_i is the highest active job and if its preemption level is strictly greater than the system preemption ceiling.

With MSRP at time t, a preemption ceiling $\rho(t, \pi_j)$ is fixed for each processor π_j. For each global resource $\mathcal{R}_{Gk}$, each processor π_j defines a preemption ceiling $\rho(\mathcal{R}_{Gk})$ higher than the highest preemption ceiling on π_j. For local resources, the MSRP behavior is the same as SRP. When a job attempts to lock a global resource $\mathcal{R}_{Gk}$ on π_j, $\rho(t, \pi_j)$ is raised to $\rho(\mathcal{R}_{Gk})$ making J_i non-preemptive.

2.3 FMLP

Flexible Multiprocessor Locking Protocol (FMLP) is a synchronization protocol exclusively developed for the multiprocessor systems. It has been proposed by Block *et al.* in [3]. Initially, FMLP has been proposed to synchronized shared resources in both partitioned and global EDF scheduling. It has been extended to deal with the partitioned Fixed-Task-Priority scheduling approach of Brandenburg and Anderson in [5].

FMLP takes advantage of both busy-wait and by suspension approaches. To do that, two types of resource are considered: short ($\mathcal{R}_k^s$) and long $\mathcal{R}_k^l$) resources. When a job J_i attempts to lock an already locked short resource $\mathcal{R}_k^s$, it busy waits non-preemptively for $\mathcal{R}_k^s$. This behavior tends to reduce response-time of jobs when they use shared resources for short time.

3. TASK MARGIN

We focus on temporal robustness which consists in tolerance to temporal faults in systems. The schedulability analysis of fault-tolerant systems has been studied by Burns, Davis and Punnekkat [6]. More recently, Bougueroua, George and Midonnet has proposed an approach to compute a margin of tolerance on the WCET [4]. We present it in Section 3.1 and we propose its extension to the domain of periods in the Section 3.2.

In this section, we use the notation C_i^* which is defined by $C_i^* = C_i + B_i$.

3.1 WCET margin

The constraints in a hard real-time system are defined such that no deadlines of any task are missed. Moreover, the WCET of a task is estimated or computed in order to ensure that the task never runs for a duration longer than its WCET. If a task commits a WCET overruns fault, the system may fail unless the task has enough *WCET margin*.

DEFINITION 1 (WCET MARGIN). *The WCET margin A_i^C of a task τ_i is the additional execution time which can be added to the WCET C_i of τ_i in such a way that no task of τ misses its deadline.*

A value A_i^C is a correct WCET margin value for τ_i on π_j if it verify the inequalities:

$$U(\pi_j) + \frac{A_i^C}{T_i} \leq 1 \tag{1}$$

$$R_i^{k+1} = C_i^* + A_i^C + \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_h} \right\rceil C_h \leq D_i \tag{2}$$

$\forall \tau_l \in lp(\tau_i)$,

$$R_l^{k+1} = C_l^* + \sum_{\tau_h \in hp(\tau_l)} \left\lceil \frac{R_l^k}{T_h} \right\rceil C_h + \left\lceil \frac{R_l^k}{T_i} \right\rceil A_i^C \leq D_l \tag{3}$$

The maximum value of WCET margin for τ_i can be computed by binary search over the interval $[0, \min(D_i - C_i, (1 - U(\pi_j))T_i)]$.

3.2 Frequency margin

The sporadic (and periodic) tasks are characterized by a minimum inter-arrival time. If a job is activated as earlier as planed (e.g. faulty external component), the system may fail unless the task has enough *frequency margin*.

DEFINITION 2 (FREQUENCY MARGIN). *The frequency margin A_i^f of a task τ_i is a period of time which can be subtracted to the period T_i of τ_i in such a way that no task of τ misses its deadline.*

A value A_i^f is a correct frequency margin value for τ_i on π_j if it verify the inequalities:

$$U(\pi_j) - U_i + \frac{C_i}{T_i - A_i^f} \leq 1 \tag{4}$$

$$R_i^{k+1} = C_i^* \sum_{\tau_h \in hp(\tau_i)} \left\lceil \frac{R_i^k}{T_h} \right\rceil C_h \leq T_i - A_i^f \tag{5}$$

$\forall \tau_l \in lp(\tau_i)$

$$R_l^{k+1} = C_l^* + \sum_{\tau_h \in hp(\tau_l) - \tau_i} \left\lceil \frac{R_l^k}{T_h} \right\rceil C_h + \left\lceil \frac{R_l^k}{T_i - A_i^f} \right\rceil C_i \leq D_l \tag{6}$$

The maximum value of frequency margin for τ_i can be computed by binary search over the interval $[0, T_i - 1]$.

3.3 Margin of dependent tasks

For the case of independent tasks, if a fault occurs during the execution of a job J_i, J_i can continue its execution for A_i^C. If the tasks share resources, the WCET margin of J_i in a critical section can be less than A_i^C. In Figure 1(a), we represent the schedule of 4 jobs characterized by the parameters given in Table 1. The synchronization protocol is FMLP and the resources are considered as long. We consider each job executing for its WCET ($e_i = C_i$). If a fault occurs at the end of the execution of J_3, it can continue until time 16 in such a way that no job misses its deadline. Then the value of WCET margin A_i^C is 5 units of time if it is used at the end of the execution of J_3.

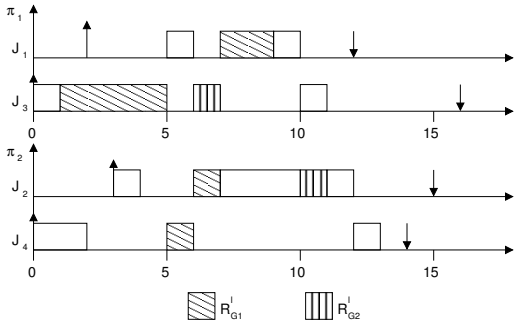
Job	Proc.	r_i	e_i	d_i
J_1	π_1	2	$1 + 2(\mathcal{R}_{G1}^l) + 1$	12
J_2	π_2	3	$1 + 1(\mathcal{R}_{G1}^l) + 3 + 1(\mathcal{R}_{G2}^l) + 1$	15
J_3	π_1	0	$1 + 5(\mathcal{R}_{G1}^l) + 1(\mathcal{R}_{G2}^l) + 1$	16
J_4	π_2	0	$2 + 1(\mathcal{R}_{G1}^l) + 1$	14

Table 1: Parameters of jobs in Figure 1.

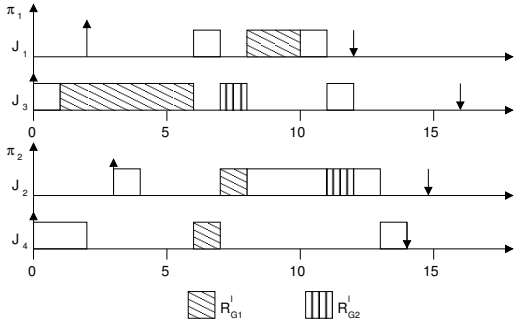
In Figure 1(b), we represent an execution overrun in the critical section associated to $\mathcal{R}_{G1}^l$. J_3 can increase its budget of time by just 1 unit of time. We notice that the WCET margin of a job in a critical section is bounded by the minimum margin of all the jobs which share the resource guarded by this section.

4. ROBUST PARTITIONING

Simulated annealing is a generic algorithm which has been firstly proposed by Kirkpatrick, Gelatt and Vecchi [12] for the optimization problems. The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of



(a) WCET margin outside critical section.



(b) WCET margin inside critical section.

Figure 1: WCET margin of dependent tasks.

its crystals and to reduce their defects. In this work, we apply the simulated annealing technique to build an algorithm which finds a feasible partition of a set of tasks where the robustness to the WCET overruns faults is maximized. We describe this algorithm, *Robust Simulated Annealing* (RPSA) in Algorithm 1. The initialization is made as follows. At line 1, the function `random_partition()` build a partition P by allocating each of the n tasks on one of the m processors randomly. This partition may be unfeasible. At line 2, an initial temperature is computed such as 99% of the partitions are kept even if they do not improve the solution. By cooling the system (decreasing the temperature), the unsatisfying solutions will be eliminated. At line 3, we initialize the `max_try` value to an integer which depends both of the number of tasks and of the number of processors. The loop at line 4-21 performs `max_try` iterations of the loop at line 6-19. After each iteration, the system is cooled by dividing the temperature by 2. The main part of the algorithm is the loop at line 6-19. At line 7, the function `compute_energy()` computes the energy of the partition P. This function is more detailed later in Algorithm 2. At line 8, a partition P_n which is the neighbor of the partition P is computed. This partition P_n is obtained either by randomly swapping two tasks of P or by randomly moving a task of P from a processor to another. The energy of this new partition is computed at line 9. If the value E_n of the energy of P_n is less than the value E_p of the energy of P then P is replaced by P_n. Otherwise, a random number is drawn between 0 and 1. The more the temperature *temp* high is, the more the probability that the value e^x ($x = \frac{E_p - E_n}{temp}$) is greater than the random number is. If $e^x > random(0,1)$ then P is also replaced by P_n else P_n is discarded. This behavior avoids that the energy converge to a local minimum. We

Algorithm 1: RPSA

```
1  P = random_partition(n, m);
2  temp = -m / ln(0.99);
3  max_try = n · m;
4  while temp > 10^-5 do
5      k = 0;
6      while k ≠ max_try do
7          E_p = compute_energy(P);
8          P_n = neighbor(P);
9          E_n = compute_energy(P_n);
10         if E_n < E_p then
11             P = P_n;
12         else
13             x = (E_p - E_n) / temp;
14             if e^x ≥ random(0,1) then
15                 P = P_n;
16             end
17         end
18         k = k + 1;
19     end
20     temp = temp / 2;
21 end
```

now describe the function `compute_energy()`. The aim of this function is to compute a value of energy for a partition such that the more the minimum value of margin for the system is great, the less the value of energy is. The value of energy is computed as follows. For each processor π_j, if the processor is empty then the value of energy is increased by 1. This behavior increase the probability that the tasks are well distributed among the processor and no processors stay empty. If the set of tasks allocated on this processor is unschedulable then the value of energy is also increased by 1 to eliminate the unfeasible partitions. For each processor π_j where the set of tasks is schedulable, the sum of margin on the execution duration values of each task is stored in the array `margin` at index j. At the end of the loop at line 3-10, the value of energy is increased by the sum of all the values stored in the array `margin`. Consequently the more the value of margin of each task great is, the less the value of energy is.

Algorithm 2: compute_energy(P)

```
1  energy = 0;
2  margin[m];
3  foreach π_j ∈ P do
4      if π_j is empty or π_j is unschedulable then
5          energy = energy + 1;
6          margin[j] = 0;
7      else
8          margin[j] = Σ_{τ_i ∈ τ(π_j)} A_i
9      end
10 end
11 energy = energy + 1 / Σ_{k=1}^{m} margin[k]
```

5. SIMULATION

5.1 Methodology

We implemented a simulator of real-time systems which provides several partitioning algorithms. Among others, it implements both RPSA and SPA algorithms.

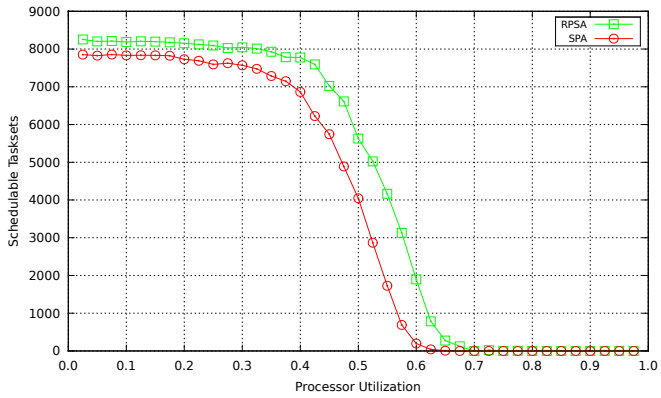
We also implemented a taskset generator which randomly makes sets of tasks with shared resources. The tasks generation process is based on the *UUniFast-Discard* algorithm proposed by Davis and Burns [7]. This algorithm is the extension to the multiprocessor case of the *UUniFast* algorithm proposed by Bini and Buttazzo [2].

For each value of processor utilization in $[0.025, 0.05, \ldots, 0.975]$, we randomly generated 1,000 tasksets with constrained deadlines. Each simulation consists in the partitioning of a taskset of 16 tasks on 4 processors. Each task has a random number of critical section in $\{0, 1, 2\}$. There are half as many resources as critical sections.

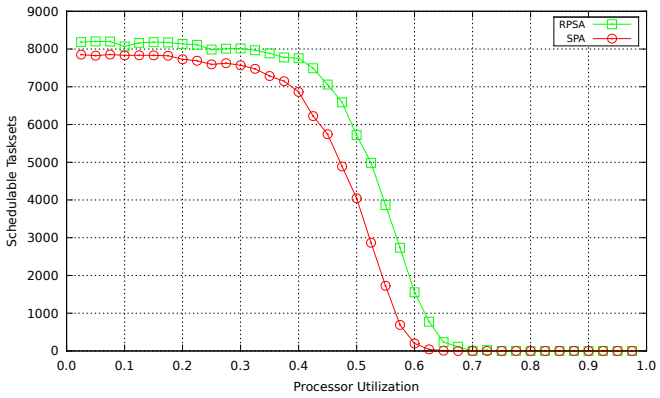
SPA has been initially proposed in order to use MPCP as synchronization protocol. In the context of our simulations, we have modified it to be based on FMLP.

5.2 Results

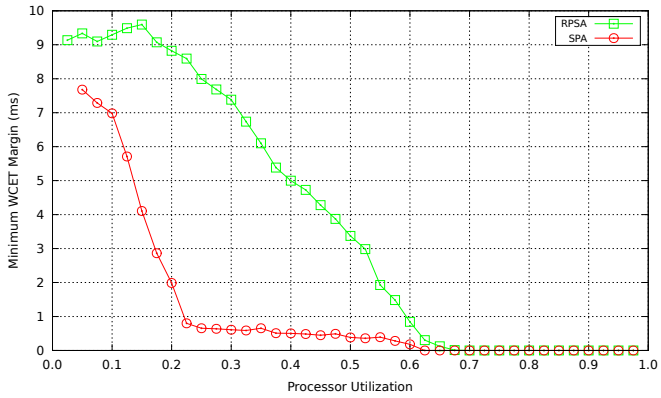
In Figure 2, we represent the comparison between RPSA and SPA in terms of schedulability and minimum margin. In Figures 2(a) and 2(c), RPSA has been implemented to maximize the WCET margin. The `compute_energy()` function of RPSA uses the Equations (1), (2) and (3) (described



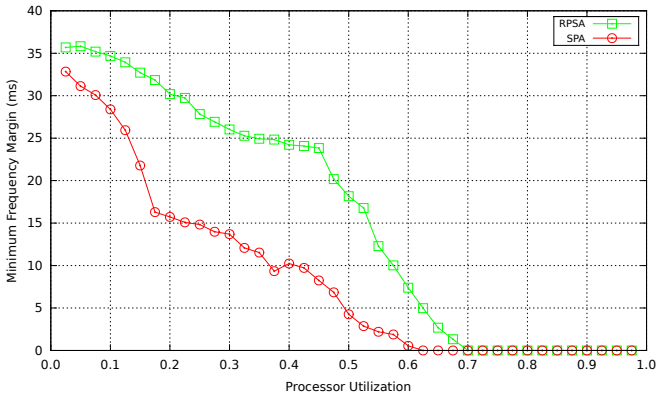
(a) Schedulability of RPSA (WCET margin) vs SPA.



(b) Schedulability of RPSA (Frequency margin) vs SPA.



(c) Minimum margin of RPSA (WCET margin) vs SPA.



(d) Minimum margin of RPSA (Frequency margin) vs SPA.

Figure 2: Simulations of RPSA and SPA on 4 processors.

in Section 3.1). In Figures 2(b) and 2(d), it has been implemented using the Equations (4), (5) and (6) (described in Section 3.2).

We show that RPSA outperforms on average SPA in terms of schedulability. Our algorithm takes advantage of the *simulated annealing* technique. Then it find solutions which can not be found by the BF algorithm (the one on which SPA is based).

In terms of robustness, RPSA produces partitions for which the margin of the tasks is increased compared to SPA.

6. CONCLUSION

We have proposed the partitioning algorithm RPSA based on the *simulated annealing* technique. We have considered a model of sporadic tasks with shared resources. The synchronization of data is performed by the FMLP protocol. Our algorithm allocates the tasks on the processors in order to maximize the robustness to the WCET overruns faults and MIT violations. We have implemented our algorithm in our simulator of real-time systems and we compare it with the SPA heuristic. We have shown that our solution outperforms the heuristic approach in terms of both schedulability and robustness.

7. REFERENCES

[1] T. P. Baker. Stack-based scheduling of realtime processes. *Real-Time Systems*, 3(1):67–99, March 1991.

[2] E. Bini and G. C. Buttazzo. Biasing effects in schedulability measures. In *Proceedings of the 16th Euromicro Conference on Real-time Systems (ECRTS)*, pages 196–203, Catania, Sicily, Italy, June - July 2004. IEEE Computer Society.

[3] A. D. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson. A flexible real-time locking protocol for multiprocessors. In *Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 47–56, Daegu, South Korea, August 2007. IEEE Computer Society.

[4] L. Bougueroua, L. George, and S. Midonnet. Dealing with execution-overruns to improve the temporal robustness of real-time systems scheduled FP and EDF. In *Proceedings of the 2nd International Conference on Systems (ICONS)*, page 8pp, Sainte-Luce, Martinique, April 2007. IEEE Computer Society.

[5] B. B. Brandenburg and J. H. Anderson. An implementation of the PCP, SRP, D-PCP, M-PCP, and FMLP real-time synchronization protocols in $LITMUS^{RT}$. In *Proceedings of the 14th IEEE International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA)*, pages

185–194, Kaohsiung, Taiwan, August 2008. IEEE Computer Society.

[6] A. Burns, R. I. Davis, and S. Punnekkat. Feasibility anlaysis of fault-tolerant real-time task sets. In *Proceedings of the 8th Euromicro Workshop on Real-Time Systems (EWRTS)*, pages 29–33, L'Aquila, Italy, June 1996. IEEE Computer Society.

[7] R. I. Davis and A. Burns. Improved priority assignment for global fixed priority pre-emptive scheduling in multiprocessor real-time systems. *Real-Time Systems*, 47(1):1–40, 2010.

[8] M. Di Natale and J. A. Stankovic. Applicability of simulated annealing methods to real-time scheduling and jitter control. In *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS)*, pages 190–199, Pisa, Italy, December 1995. IEEE Computer Society.

[9] K. Echtle and I. Eusgeld. A genetic algorithm for fault-tolerant system design. In R. de Lemos, T. Weber, and J. Camargo, editors, *Dependable Computing*, volume 2847, pages 197–213. Springer Berlin / Heidelberg, 2003.

[10] P. Emberson and I. Bate. Extending a task allocation algorithm for graceful degradation of real-time distributed embedded systems. In *Proceedings of the 29th IEEE Real-Time Systems Symposium (RTSS)*, pages 270–279, Barcelona, Spain, November-December 2008. IEEE Computer Society.

[11] P. Gai, M. Di Natale, G. Lipari, A. Ferrari, C. Gabellini, and P. Marceca. A comparison of MPCP and MSRP when sharing resources in the Janus multiple-processor on a chip platform. In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 189–198, Toronto, Canada, May 2003. IEEE Computer Society.

[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[13] Y. Oh and S. H. Son. Scheduling real-time tasks for dependability. *The Journal of the Operational Research Society*, 46(6):629–639, June 1997.

[14] X. Qin and H. Jiang. A novel fault-tolerant scheduling algorithm for precedence constrained tasks in real-time heterogeneous systems. *Parallel Computing*, 32(5-6):331–356, June 2006.

[15] R. (Raj) Rajkumar. Real-time synchronization protocols for shared memory multiprocessors. In *Proceedings of the 10th International Conference on Distributed Computing Systems (ICDCS)*, pages 116–123, Paris, France, May-June 1990. IEEE Computer Society.

[16] L. Sha, R. Rajkumar, and J. P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, September 1990.

[17] K. Tindell, A. Burns, and A. J. Wellings. Allocating hard real-time tasks: An NP-hard problem made easy. *Real-Time Systems*, 4(2):145–165, June 1992.